CS-473 Assignment #1

Deadline: Monday, 16\textsuperscript{th} December, 2002

Book: Network Security: Private Communication in a Public World

Exercise: 2.7
Questions: #3, #5
Pages: 57–58

The relevant pages from the Book are attached.
2.4.2 Transmitting Over an Insecure Channel

It is often impossible to prevent eavesdropping when transmitting information. For instance, a telephone conversation can be tapped, a letter can be intercepted, and a message transmitted on a LAN can be received by unauthorized stations.

If you and I agree on a shared secret (a key), then by using secret key cryptography we can send messages to one another on a medium that can be tapped, without worrying about eavesdroppers. All we need to do is have the sender encrypt the messages and the receiver decrypt them using the shared secret. An eavesdropper will only see unintelligible data.

This is the classic use of cryptography.

2.4.3 Secure Storage on Insecure Media

If I have information I want to preserve but which I want to ensure no one else can look at, I have to be able to store the medium where I am sure no one can get it. Between clever thieves and court orders, there are very few places that are truly secure, and none of these is convenient. If I invent a key and encrypt the information using the key, I can store it anywhere and it is safe so long as I can remember the key. Of course, forgetting the key makes the data irrevocably lost, so this must be used with great care.

2.4.4 Authentication

In spy movies, when two agents who don’t know each other must rendezvous, they are each given a password or pass phrase that they can use to recognize one another. This has the problem that anyone overhearing their conversation or initiating one falsely can gain information useful for replaying later and impersonating the person to whom they are talking.

The term strong authentication means that someone can prove knowledge of a secret without revealing it. Strong authentication is possible with cryptography. Strong authentication is particularly useful when two computers are trying to communicate over an insecure network (since few people can execute cryptographic algorithms in their heads). Suppose Alice and Bob share a key $K_{AB}$ and they want to verify they are speaking to each other. They each pick a random number, which is known as a challenge. Alice picks $r_A$, Bob picks $r_B$. The value $x$ encrypted with the key $K_{AB}$ is known as the response to the challenge $x$. How Alice and Bob use challenges and responses to authenticate each other is shown in Figure 2-1.

If someone, say Fred, were impersonating Alice, he could get Bob to encrypt a value for him (though Fred wouldn’t be able to tell if the person he was talking to was really Bob), but this information would not be useful later in impersonating Bob to the real Alice because the real Alice
would pick a different challenge. If Alice and Bob complete this exchange, they have each proven to the other that they know $K_{AB}$ without revealing it to an impostor or an eavesdropper. Note that in this particular protocol, there is the opportunity for Fred to obtain some (chosen plaintext, ciphertext) pairs, since he can claim to be Bob and ask Alice to encrypt a challenge for him. For this reason, it is essential that challenges be chosen from a large enough space, say $2^{64}$ values, so that there is no significant chance of using the same one twice.

That is the general idea of a cryptographic authentication algorithm, though this particular algorithm has a subtle problem that would prevent it from being useful in most computer-to-computer cases. (We would have preferred not bringing that up, but felt we needed to say that so as not to alarm people who already know this stuff and who would realize the protocol was not secure. Protocol flaws such as this, and methods of fixing them, are discussed in Chapter 11 Security Handshake Pitfalls.)

### 2.4.5 Integrity Check

A secret key scheme can be used to generate a fixed-length cryptographic checksum associated with a message. This is a rather nonintuitive use of secret key technology.

What is a checksum? An ordinary (noncryptographic) checksum protects against accidental corruption of a message. The original derivation of the term checksum comes from the operation of breaking a message into fixed-length blocks (for instance, 32-bit words) and adding them up. The sum is sent along with the message. The receiver similarly breaks up the message, repeats the addition, and checks the sum. If the message had been garbled en route, the sum will not match the sum sent and the message is rejected, unless, of course, there were two or more errors in the transmission that canceled one another. It turns out this is not terribly unlikely, given that if flaky hardware turns a bit off somewhere, it is likely to turn a corresponding bit on somewhere else. To protect against such "regular" flaws in hardware, more complex checksums called CRCs were devised. But these still only protect against faulty hardware and not an intelligent attacker. Since CRC algorithms are published, an attacker who wanted to change a message could do so, compute the CRC on the new message, and send that along.
2.6.4 Downline Load Security

It is common practice to have special-purpose devices connected to a network, like routers or printers, that do not have the nonvolatile memory to store the programs they normally run. Instead, they keep a bootstrap program smart enough to get a program from the network and run it. This scheme is called **downline load**.

Suppose you want to downline load a program and make sure it hasn’t been corrupted (whether intentionally or not). If you know the proper hash of the program, you can compute the hash of the loaded program and make sure it has the proper value before running the program.

2.6.5 Digital Signature Efficiency

The best-known public key algorithms are sufficiently processor-intensive that it is desirable to compute a message digest of the message and sign that, rather than to sign the message directly. The message digest algorithms are much less processor-intensive, and the message digest is much shorter than the message.

2.7 HOMEWORK

1. What is the dedication to this book?

2. Random J. Protocol-Designer has been told to design a scheme to prevent messages from being modified by an intruder. Random J. decides to append to each message a hash of that message. Why doesn’t this solve the problem? (We know of a protocol that uses this technique in an attempt to gain security.)

3. Suppose Alice, Bob, and Carol want to use secret key technology to authenticate each other. If they all used the same secret key \( K \), then Bob could impersonate Carol to Alice (actually any of the three can impersonate the other to the third). Suppose instead that each had their own secret key, so Alice uses \( K_A \), Bob uses \( K_B \), and Carol uses \( K_C \). This means that each one, to prove his or her identity, responds to a challenge with a function of his or her secret key and the challenge. Is this more secure than having them all use the same secret key \( K \)? (Hint: what does Alice need to know in order to verify Carol’s answer to Alice’s challenge?)

4. As described in §2.6.4 Downline Load Security, it is common, for performance reasons, to sign a message digest of a message rather than the message itself. Why is it so important that it be difficult to find two messages with the same message digest?
5. What's wrong with the protocol in §2.4.4 Authentication? (Hint: assume Alice can open two connections to Bob.)

6. Assume a cryptographic algorithm in which the performance for the good guys (the ones that know the key) grows linearly with the length of the key, and for which the only way to break it is a brute-force attack of trying all possible keys. Suppose the performance for the good guys is adequate (e.g., it can encrypt and decrypt as fast as the bits can be transmitted over the wire) at a certain size key. Then suppose advances in computer technology make computers twice as fast. Given that both the good guys and the bad guys get faster computers, does this advance in computer speed work to the advantage of the good guys, the bad guys, or does it not make any difference?